Learning and Tuning Fuzzy Logic Controllers Through Reinforcements

Hamid Berenji
Sterling Software
AI Research Branch, Mail Stop 269-2
NASA Ames Research Center
Moffett Field, CA 94025

Pratap Khedkar
EECS Department
University of California
Berkeley, CA 94720

NASA Ames Research Center
Artificial Intelligence Research Branch

Technical Report FIA-92-02

January, 1992

FIA-92-01

TDAG: An Algorithm for Learning to Predict Discrete Sequences

Phil Laird

January 1992

Learning from experience to predict sequences of discrete symbols is a fundamental problem in machine learning with many applications. We apply sequence prediction using a simple and practical sequence-prediction algorithm, called TDAG. The TDAG algorithm is first tested by comparing its performance with some common data compression algorithms. Then it is adapted to the detailed requirements of dynamic program optimization, with excellent results.

FIA-92-02

Learning and Tuning Fuzzy Logic Controllers Through Reinforcements

Hamid Berenji and Pratrp Khedkar

January 1992

This paper presents a new method for learning and tuning a fuzzy logic controller based on reinforcements from a dynamic system. In particular, our Generalized Approximate Reasoning-based Intelligent Control [GARIC] architecture [a] learns and tunes a fuzzy logic controller even when only weak reinforcements, such as a binary failure signal, is available; [b] introduces a new conjunction operator in computing the rule strengths of fuzzy control rules; [c] introduces a new localized mean of maximum [LMOM] method in combining the conclusions of several firing control rules; and [d] learns to produce real-valued control actions. Learning is achieved by integrating fuzzy inference into a feedforward network, which can then adaptively improve performance by using gradient descent methods. We extend the AHC algorithm of Barto, Sutton, and Anderson to include the prior control knowledge of human operators. The GARIC architecture is applied to a cart-pole balancing system and has demonstrated significant improvements in terms of the speed of learning and robustness to changes in the dynamic system's parameters over previous schemes for cart-pole balancing.

FIA-92-03

Smadar Kedar

January 1992

FIA-92-04

There is No Free Lunch: Tradeoffs in the Utility of Learned Knowledge

Smadar Kedar and Kate McKusick

March 1992

Planning systems which make use of domain theories can produce more accurate plans and achieve more goals as the quality of their domain knowledge improves. MTR, a multi-strategy learning system, was designed to learn from system failures and improve domain knowledge used in planning. However, augmented domain knowledge can decrease planning efficiency. We describe how improved knowledge that becomes expensive to use can be approximated to yield calculated tradeoffs in accuracy and efficiency.

Learning and Tuning Fuzzy Logic Controllers Through Reinforcements

Hamid R. Berenji
Sterling Software
Artificial Intelligence Research Branch
NASA Ames Research Center
MS: 244-17, Mountain View, CA 94035
e-mail: berenji@ptolemy.arc.nasa.gov

Pratap Khedkar
EECS Department
University of California, Berkeley
Berkeley, CA 94720
email: khedkar@janus.berkeley.edu

Abstract

This paper presents a new method for learning and tuning a fuzzy logic controller based on reinforcements from a dynamic system. In particular, our Generalized Approximate Reasoning-based Intelligent Control (GARIC) architecture (a) learns and tunes a fuzzy logic controller even when only weak reinforcements, such as a binary failure signal, is available; (b) introduces a new conjunction operator in computing the rule strengths of fuzzy control rules; (c) introduces a new localized mean of maximum (LMOM) method in combining the conclusions of several firing control rules; and (d) learns to produce real-valued control actions. Learning is achieved by integrating fuzzy inference into a feedforward network, which can then adaptively improve performance by using gradient descent methods. We extend the AHC algorithm of Barto, Sutton, and Anderson to include the prior control knowledge of human operators. The GARIC architecture is applied to a cart-pole balancing system and has demonstrated significant improvements in terms of the speed of learning and robustness to changes in the dynamic system's parameters over previous schemes for cart-pole balancing.

1 Introduction

The non-linear behavior of many practical systems and unavailability of quantitative data regarding the input-output relations makes the analyti-

cal modeling of these systems very difficult. On the other hand, approximate reasoning-based controllers which do not require analytical models have demonstrated a number of successful applications such as the subway system in the city of Sendai [31], nuclear reactor control [12] and automobile transmission control [14]. These applications have mainly concentrated on emulating the performance of a skilled human operator in the form of linguistic rules. However, the process of learning and tuning the control rules to achieve the desired performance remains a difficult task.

Starting with the Self Organizing Control (SOC) techniques of Mamdani and his students (e.g., [23]), the need for research in developing fuzzy logic controllers which can learn from experience has been realized (e.g., [17]). The learning task may include the identification of the main control parameters (better known as *system identification* in control theory) or development and tuning of the fuzzy memberships used in the control rules. In this paper, we concentrate on the latter learning task and develop an architecture which can learn to adjust the fuzzy membership functions of the linguistic labels used in different control rules.

Connectionist learning approaches [5] can be used in learning control. Here, we can distinguish three classes: *supervised learning*, *reinforcement learning*, and *unsupervised learning*. In supervised learning, a teacher provides the desired control objective at each time step to the learning system. In reinforcement learning, the teacher's response is not as direct, immediate, and informative as in supervised learning and it serves more to evaluate the state of the system. The presence of a teacher or a supervisor to provide the correct control response is not assumed in unsupervised learning.

If supervised learning can be used in control (e.g., when the input-output training data is available), it has been shown that it is more efficient than reinforcement learning (e.g., [6, 1]). However, many control problems require selecting control actions whose consequences emerge over uncertain periods for which input-output training data are not readily available. In such domains, reinforcement learning techniques are more appropriate than supervised learning.

The organization of this paper is as follows. We first review some fundamentals of fuzzy logic control, reinforcement learning, and credit assignment. Next, we discuss the general architecture for Approximate Reasoning-based Intelligent Control (GARIC). This architecture addresses two related

problems. First, we introduce techniques for the design of rule-based controllers which use qualitative linguistic rules obtained from human expert controllers. Also, we describe a controller that learns directly from experience and automatically develops and adjusts the definitions of its linguistic labels. Finally, we describe the application of this architecture to the real-world control problem of cart-pole balancing.

2 Fuzzy Sets and Fuzzy Logic Control

A fuzzy set, defined originaly by Zadeh [32], is an extension of a crisp set. Crisp sets only allow full membership or no membership at all, whereas fuzzy sets allow partial membership. In other words, an element may partially belong to a set. In a crisp set, the membership or non-membership of an element x in set A is described by a characteristic function $\mu_A(x)$, where:

$$\mu_A(x) = \begin{cases} 1 & \text{if } x \in A \\ 0 & \text{if } x \notin A. \end{cases}$$

Fuzzy set theory extends this concept by defining *partial memberships* which can take values ranging from 0 to 1:

$$\mu_A : X \rightarrow [0, 1]$$

where X refers to the universal set defined in a specific problem.

Assuming that A and B are two fuzzy sets with membership functions of μ_A and μ_B, then the following operations can be defined on these sets. The *complement* of a fuzzy set A is a fuzzy set $\bar{A}$ with a membership function

$$\mu_{\bar{A}} = 1 - \mu_A(x).$$

The *union* of A and B is a fuzzy set with the following membership function

$$\mu_{A \cup B} = \max\{\mu_A, \mu_B\}.$$

The *intersection* of A and B is a fuzzy set

$$\mu_{A \cap B} = \min\{\mu_A, \mu_B\}.$$

Different methods for developing fuzzy logic controllers have been suggested in recent years and are reviewed in [8]. In the design of a fuzzy controller, one must identify the main control parameters and determine a term set which is at the right level of granularity for describing the values of each linguistic variable*. For example, a term set including linguistic values such as { *Small, Medium, Large*} may not be satisfactory in some domains, and may instead require the use of a five term set such as {*Very Small, Small, Medium, Large, and Very Large*}.

Figure 1 illustrates a simple architecture for a fuzzy logic controller. The system dynamics of the plant is measured by a set of sensors. This architecture consists of four elements whose functions are described next.

In coding the values from the sensors, one transforms the values of the sensor measurements by using the linguistic labels in the rule preconditions. This process is commonly called *fuzzification* or *encoding*. The fuzzification stage requires matching the sensor measurements against the membership functions of linguistic labels.

In modeling the human expert operator's knowledge, fuzzy control rules of the form:

IF Error is small AND Change-in-error is small THEN Force is small

can be used effectively when expert human operators can express the heuristics or the control knowledge that they use in controlling a process in terms of rules of the above form.

2.1 Conflict Resolution and Decision Making

As mentioned earlier, due to the partial matching attribute of fuzzy control rules and the fact that the preconditions of rules do overlap, more than one fuzzy control rule can fire at a time. The methodology which is used in deciding what control action should be taken as the result of the firing of several rules can be referred to as *conflict resolution*. The following example, using two rules, illustrates this process. Assume that we have the following rules:

*A linguistic variable is a variable which can only take linguistic values.

Rule 1: IF X is A_1 and Y is B_1 THEN Z is C_1
Rule 2: IF X is A_2 and Y is B_2 THEN Z is C_2

Each rule has an *antecedent* or *if* part containing several preconditions, and a *consequent* or *then* part which prescribes the value of one or more output actions. Now, if we have x_0 and y_0 as the sensor readings for fuzzy variables X and Y, then their *truth values* are represented by $\mu_{A_1}(x_0)$ and $\mu_{B_1}(y_0)$ respectively for Rule 1, where μ_{A_1} and μ_{B_1} represent the membership function for A_1 and B_1, respectively. Similarly for Rule 2, we have $\mu_{A_2}(x_0)$ and $\mu_{B_2}(y_0)$ as the truth values of the preconditions.

$$w_1 = \wedge(\mu_{A_1}(x_0), \mu_{B_1}(y_0))$$

Similarly for Rule 2:

$$w_2 = \wedge(\mu_{A_2}(x_0), \mu_{B_2}(y_0)).$$

where $\wedge$ denotes a conjunction or intersection operator. Traditionally, fuzzy logic controllers use a *minimum* operator for $\wedge$. However, here we use a *softmin* operator which produces the same result in the limit but in general is not as specific as the *minimum* operator is. The reason for this is differentiability, which we need for learning purposes. This will be dealt with in greater detail later.

Using the softmin, the *strength* of Rule 1 can be calculated by:

$$w_1 = \frac{\mu_{A_1}(x_0)e^{-k\mu_{A_1}(x_0)} + \mu_{B_1}(y_0)e^{-k\mu_{B_1}(y_0)}}{e^{-k\mu_{A_1}(x_0)} + e^{-k\mu_{B_1}(y_0)}}$$

Similarly for Rule 2:

$$w_2 = \frac{\mu_{A_2}(x_0)e^{-k\mu_{A_2}(x_0)} + \mu_{B_2}(y_0)e^{-k\mu_{B_2}(y_0)}}{e^{-k\mu_{A_2}(x_0)} + e^{-k\mu_{B_2}(y_0)}}$$

The control output of rule 1 is calculated by applying the matching strength of its preconditions on its conclusion. Assuming that

$$z_1 = \mu_{c_1}^{-1}(w_1),$$

and for Rule 2:

$$z_2 = \mu_{c_2}^{-1}(w_2),$$

In this paper, we introduce a new defuzzification procedure to compute the expression $\mu^{-1}(w)$ which is explained later. The above equations show that as a result of reading sensor values x_0 and y_0, Rule 1 is recommending a control action z_1 and Rule 2 is recommending a control action z_2. The combination of the above rules produces a nonfuzzy control action z^* which is calculated using a weighted averaging approach:

$$z^* = \frac{\sum_{i=1}^{n} w_i z_i}{\sum_{i=1}^{n} w_i}$$

where n is the number of rules, and z_i is the amount of control action recommended by rule i. A similar procedure can be used for multiple output variables in the consequents.

3 Reinforcement Learning

In reinforcement learning, one assumes that there is no supervisor to critically judge the chosen control action at each time step. The learning system is told indirectly about the effect of its chosen control action. The study of reinforcement learning relates to *credit assignment* where, given the performance (results) of a process, one has to distribute reward or blame to the individual elements contributing to that performance. This may be further complicated if there is a sequence of actions, which is collectively awarded a delayed reinforcement. In rule-based systems, for example, this means assigning credit or blame to individual rules (or their parts) engaged in the problem solving process. Samuel's checkers-playing program is probably the earliest AI program which used this idea [25]. Michie and Chambers [19] used a reward-punishment strategy in their BOXES system, which learned to do cart-pole balancing by discretizing the state space into *non-overlapping* regions (boxes) and applying two opposite constant forces. Barto, Sutton, and Anderson [4] used two neuron-like elements to solve the learning problem in cart-pole balancing. In these approaches, the state-space is partitioned into non-overlapping smaller regions and then the credit assignment is performed on a local basis.

Reinforcement learning has its roots in studies of animal learning and research on human behavior (e.g., [3]). It directly relates to the theory of *learning automata* initiated by the work of Tsetlin [28] and further devel-

oped by the work of Narendra and Thathachar [22], Narendra and Lakshmivarahan [21] , and Mendel and McLaren [18] in control engineering. Since reinforcement learning techniques do not use an explicit teacher or supervisor, they construct an internal evaluator or a *critic* capable of evaluating the dynamic system's performance. The construction of this critic so that it can properly evaluate the performance in a way which is useful to the control objective, is itself a significant problem in reinforcement learning. Given the evaluation by the critic, the other problem in reinforcement learning is how to adjust the control signal. Barto [5] discusses several approaches to this problem based on the gradient of the critic's evaluation as a function of control signals.

Temporal Difference methods Related to reinforcement learning are the Temporal Difference (TD) methods, a class of incremental learning procedures specialized for prediction problems, which have been introduced by Sutton [27]. The main characteristic of these methods is that they learn from successive predictions whereas in the case of supervised learning methods, learning occurs when the difference between the predicted outcome and the actual outcome is revealed (i.e., the learning model in TD does not have to wait until the actual outcome is known and can update its parameters *within* a trial period). The difference between the Temporal Difference methods and the supervised learning methods becomes clear when these methods are distinguished as single-step versus multi-step prediction problems. In the single-step prediction (e.g., Widrow-Hoff rule [29]), complete information regarding the correctness of a prediction is revealed at once. However, in multi-step prediction, this information is not revealed until more than one step after the prediction is made, but partial information becomes available at each step. Barto et. al. have recently shown stronger relation between a specific class of these methods called *TD algorithm* and dynamic programming [7].

ARIC Architecture The Approximate Reasoning-based Intelligent Control (ARIC) architecure has been proposed in [10]. This architecture extends Anderson's method [1] by including the prior control knowledge of expert operators in terms of fuzzy control rules. In ARIC, a neural network is used to perform action and state evaluations. Also, two coupled neural networks are used to select a control action at each time step where the first network

uses fuzzy inference to recommend an action and the second network calculates a degree to which the action recommended by the first network should be modified. The ARIC architecture tunes its fuzzy controller through updating the weights on the links in these networks. As this learning proceeds, the action recommended by the fuzzy controller is followed more often. Only monotonic membership functions are used in ARIC and the fuzzy labels used in the control rules are adjusted locally within each rule. However, in the architecture presented next, we provide an algorithm to tune the fuzzy labels globally in all the rules and allow any type of differentiable membership function to be used in the construction of a fuzzy logic controller.

4 The GARIC architecture

Our system will determine a control action by using a neural network which implements fuzzy inference. In this way, prior expert knowledge can be easily incorporated. This knowledge is allowed to be faulty or damaged. Another neural net will learn to become a good evaluator of the current state and will serve as an internal critic. Both networks will adapt their weights concurrently so as to improve performance.

The architecture of GARIC is schematically shown in Figure 3.

It has three components:

- The Action Selection Network (ASN) maps a state vector into a recommended action F, using fuzzy inference.
- The Action Evaluation Network (AEN) maps a state vector and a failure signal into a scalar score which indicates state goodness. This is also used to produce internal reinforcement $\hat{r}$.
- The Stochastic Action Modifier (SAM) uses both F and $\hat{r}$ to produce an action F' which is applied to the plant.

The ensuing state is fed back into the controller, along with a boolean failure signal. Learning occurs by fine-tuning of the free parameters in the two networks : in the AEN, the weights are adjusted; in the ASN, the parameters describing the fuzzy membership functions change.

4.1 The Action Evaluation Network

The AEN plays the role of an adaptive critic element (ACE) [4] and constantly predicts reinforcements associated with different input states. The only information received by the AEN is the state of the physical system in terms of its state variables and whether or not a failure has occurred.

The AEN is a standard two-layer feedforward net with sigmoids everywhere except in the output layer. The input is the state of the plant, and the output is an evaluation of the state (a score), denoted by v. This v-value is suitably discounted and combined with the external failure signal to produce internal reinforcement $\hat{r}$ as explained before.

The structure of an evaluation network includes h hidden units and n input units from the environment, and a bias unit (i.e., $x_0, x_1, ..., x_n$). In this network, each hidden unit receives $n+1$ inputs and has $n+1$ weights, while each output unit receives $n+h+1$ inputs and has $n+h+1$ weights. This structure is shown in Figure 4. The learning algorithm is composed of Sutton's AHC algorithm [26] for the output unit and error back-propagation algorithm [24] for the hidden units.

The AEN produces a prediction of future reinforcement for a given state, and the changes in this prediction are used to guide the SAM in selecting actions. For example, if we move from a state with prediction of low reinforcement to a state with prediction of higher reinforcement, this positive change, also called *heuristic* or *internal reinforcement*, is used to reinforce the selection of the action which caused this move.

The output of the units in the hidden layer is:

$$y_i[t, t+1] = g(\sum_{j=1}^{n} a_{ij}[t] x_j[t+1]) \tag{1}$$

where

$$g(s) = \frac{1}{1+e^{-s}} \tag{2}$$

and t and $t+1$ are successive time steps. The output unit of the evaluation network receives inputs from both units in the hidden layer (i.e., y_i) and directly from the units in the input layer (i.e., x_i):

$$v[t, t+1] = \sum_{i=1}^{n} b_i[t] x_i[t+1] + \sum_{i=1}^{h} c_i[t] y_i[t, t+1] \tag{3}$$

where v is the prediction of reinforcement. In the above equations (and the equations which follow), double time dependencies are used to avoid instabilities in the updating of weights [2]. For example, in the above equation, the weights at time t are multiplied by the x_i's at time $t+1$. If the same time index is used, then we can not detect whether the change in v was caused by the change in the weights (i.e., b_i and c_i) or it was caused by the change in the state of the system (i.e., x_i). Writing the equation as shown above with different time steps allows us to compare different v's over times and notice whether the system has moved to a better state (i.e., higher reinforcement) or to a worse state (i.e., lower reinforcement).

This network evaluates the action recommended by the action network as a function of the failure signal and the change in state evaluation based on the state of the system at time $t+1$:

$$\hat{r}[t+1] = \begin{cases} 0 & \text{start state ;} \\ r[t+1] - v[t,t] & \text{failure state;} \\ r[t+1] + \gamma v[t,t+1] - v[t,t] & \text{otherwise} \end{cases} \quad (4)$$

where $0 \leq \gamma \leq 1$ is the *discount rate*. In other words, the change in the value of v plus the value of the external reinforcement constitutes the heuristic or internal reinforcement $\hat{r}$ where the future values of v are discounted more, the further they are from the current state of the system. For example, the value of v generated one time step later is given less weight than the the current value of v. This method of estimating reinforcement gives an approximate exponential trace of v, where the series is truncated after two terms.

4.2 Action Selection Network

Given the current state of the plant, this network selects an action by implementing an inference scheme based on fuzzy control rules as explained in section 2. It can be represented as a network with 5 layers of nodes, each layer performing one stage of the fuzzy inference process (see Figure 5). The connections are feedforward, with each node performing a local computation. However, this computation may be different from the conventional weighted-sum-of-inputs.

Layer 1 is the input layer, consisting of the real-valued input variables. These can also be thought of as the linguistic variables of interest. No computation is done at these nodes.

A **Layer 2** node corresponds to one possible value of one of the linguistic variables in Layer 1, e.g. if large is one of the values that x can take, a node computing $\mu_{large}(x)$ belongs to layer 2. It will have exactly one input, and will feed its output to all the rules using the clause: *if x is large*, in their *if* part. The function is given by

$$\mu_{c_V, s_{VL}, s_{VR}}(x)$$

where V indicates a linguistic value (e.g. *large*), and c, s_L, s_R correspond to the center, left spread and right spread of the fuzzy membership function of label V. c_V serves as a reference point (the mode), and the spreads characterize length scales on either side of the center, thus permitting asymmetry. More parameters may be included if desired. An instance of a smooth membership function is

$$\mu(x) = \frac{1}{1 + |\frac{x-c}{s}|^b}$$

where $s = s_{VL}$ or s_{VR} accordingly as $x < c$ or $x \geq c$ and b controls the curvature. For triangular shapes, this function is given by

$$\mu_{c, s_L, s_R}(x) = \begin{cases} 1 - |x - c|/s_R, & x \in [c, c + s_R] \\ 1 - |x - c|/s_L, & x \in [c - s_L, c) \\ 0 & \text{otherwise} \end{cases} \tag{5}$$

Triangular shapes are to be preferred because they are simple and have been proven to be sufficient in scores of application domains. The center and spreads may be considered as weights on the input links, analogous to the approach taken with radial-basis-function units in neural networks [20].

Layer 3 implements the conjunction of all the antecedent conditions in a rule. A node in layer 3 corresponds to a rule in the rule-base. Its inputs come from all nodes in Layer 2 which participate in the *if* part of that rule. The node itself performs the *min* operation, which we have softened to the following continuous, differentiable *softmin* operation:

$$O_{R3} = w_r = \frac{\sum_i \mu_i e^{-k\mu_i}}{\sum_i e^{-k\mu_i}} \tag{6}$$

Here, μ_i is the degree of match between a fuzzy label occurring as one of the antecedents of rule r, and the corresponding input variable. This *softmin* operation gives w_r, the degree of applicability of Rule r. The parameter k controls the hardness of the softmin operation, and as $k \to \infty$, we recover the usual *min* operator. However, for k finite, we get a differentiable function of the inputs, which makes it convenient for calculating gradients during the learning process. The choice of k is not critical.

A **Layer 4** node corresponds to a consequent label. Its inputs come from all rules which use this particular consequent label. For each of the w_r supplied to it, this node computes the corresponding output action as suggested by rule r. This mapping may be written as

$$\mu^{-1}_{c_V, s_{VL}, s_{VR}}(w_r)$$

where V indicates a specific consequent label, c, s_L, s_R parameterize the membership function as before, and the inverse is taken to mean a suitable defuzzification procedure applicable to an individual rule. In general, the mathematical inverse of μ may not exist if the function is not strictly monotonic. We propose a simple procedure to determine this inverse: if w_r is the degree to which Rule r is satisfied,

$\mu_V^{-1}(w_r)$ *is the X-coordinate of the centroid of the set* $\{x : \mu_V(x) = w_r\}$

This is similar to the Mean-of-Maximum method of defuzzification [8], but the latter is applied *after* all rule consequents have been combined, whereas we apply it locally, to each rule, *before* the consequents are combined. We will refer to this variation as the LMOM (Local Mean-of-Maximum) method[†] (see Figure 6).

For triangular functions, LMOM gives

$$\mu^{-1}_{c_V, s_{VL}, s_{VR}}(w_r) = c_V + \frac{1}{2}(s_{VR} - s_{VL})(1 - w_r) \qquad (7)$$

For the case $w_r = 0$, the limiting value of $\mu^{-1}(w_r \to 0^+)$ is used (which is $c_V + (s_{VL} + s_{VR})/2$). It is easy to see that the set $\mu^{-1}([0, 1])$ is the

[†]Although LMOM was independently derived in our work, but we were referred to Yager's level set method[30] later on by a reviewer of this paper. The LMOM and level set methods are similar in nature although Yager [30] does not discuss the case for skewed and convex fuzzy sets in any details.

projection of the median of the triangular membership function on the X-axis. If the membership function is monotonic, then $\mu^{-1}(w_r)$ is just the standard mathematical inverse, with appropriate limiting values.

The unusual feature of a unit in Layer 4 is that it may have multiple outputs carrying different values, since sharing of consequent labels is allowed. For each rule feeding it a degree, it should produce a corresponding output action which is fed to the next layer. However, this nonstandard feature can be eliminated for many classes of membership functions. For triangular functions, such a node needs to output only the value

$$O_{V4} = (c_V + \frac{1}{2}(s_{VR} - s_{VL}))(\sum_r w_r) - \frac{1}{2}(s_{VR} - s_{VL})(\sum_r w_r^2) \qquad (8)$$

In general, whenever $\mu^{-1}(x)$ is polynomial in x, only one output is sufficient, regardless of the number of inputs. This transformation is possible because of the form of the computation done in the next layer.

•Layer 5 will have as many nodes as there are output action variables. Each output node combines the recommendations from all the fuzzy control rules in the rulebase, using the following weighted sum, the weights being the rule strengths:

$$F = \frac{\sum_r w_r \mu^{-1}(w_r)}{\sum_r w_r} \qquad (9)$$

By taking advantage of the transformation used in layer 4, this may be rewritten as

$$F = \frac{\sum_V O_{V4}}{\sum_R O_{R3}} \qquad (10)$$

where the inputs come from Layer 3 and Layer 4. The node simply sums up each set of inputs and takes their quotient. This delivers a continuous output variable value which is the action selected by the ASN. F will always be defined if each dimension of the input space is completely covered by the antecedent label functions.

Modifiable weights are present on input links into Layer 2 and 4 only. The other weights are fixed at unity. This means that the gradient descent procedure effectively works on only two layers of weights, rather than all five.

4.3 Stochastic Action Modifier

This uses the values of $\hat{r}$ from the previous time step and the action F recommended by the ASN to stochastically generate an action F' which is a gaussian random variable with mean F and standard deviation $\sigma(\hat{r}(t-1))$. This $\sigma()$ is some nonnegative, monotone decreasing function, e.g. $\exp(-\hat{r})$. The action F' is what is actually applied to the plant. The stochastic perturbation in the suggested action leads to a better exploration of state space and better generalization ability. The magnitude of the deviation $|F'-F|$ is large when $\hat{r}$ is low, and small when the internal reinforcement is high. The result is that a large random step away from the recommendation results when the last action performed is bad, but the controller remains consistent with the fuzzy control rules when the previous action selected is a good one. The actual form of the function $\sigma()$, especially its scale and rate of decrease, should take the units and range of variation of the output variable into account.

The *perturbation* at each time step is denoted

$$s(t) = \frac{F'(t) - F(t)}{\sigma(\hat{r}(t-1))} \tag{11}$$

and is simply the normalized deviation from the ASN-recommended action. This will contribute as a learning factor in the ASN.

5 Learning Mechanisms

5.1 Learning in AEN

Weight-updating in this network is similar to a reward/punishment scheme for neural networks. If positive (negative) internal reinforcements are received, the values of the weights are rewarded (punished) by being changed in the direction which increases (decreases) its contribution to the total sum. The weights on the links connecting the units in the input layer directly to the units in the output layer are updated according to the following:

$$b_i[t+1] = b_i[t] + \beta\hat{r}[t+1]x_i[t] \tag{12}$$

where $\beta > 0$ is a constant and $\hat{r}[t+1]$ is the internal reinforcement at time $t+1$.

Similarly, for the weights on the connections between the hidden layer and the output layer, we have:

$$c_i[t+1] = c_i[t] + \beta \hat{r}[t+1] y_i[t,t] \tag{13}$$

The weight update function for the hidden layer is based on a modified version of the error back-propagation algorithm [24]. Since no direct error measurement is possible (i.e., knowledge of correct action is not available), as in Anderson [1], $\hat{r}$ plays the role of an error measure in the update of the output unit's weights: if $\hat{r}$ is positive, the weights are altered so as to increase the output v for positive input, and vice versa. Therefore, the equation for updating the weights is

$$a_{ij}[t+1] = a_{ij}[t] + \beta_h \hat{r}[t+1] y_i[t,t](1 - y_i[t,t]) \mathrm{sgn}(c_i[t]) x_j[t] \tag{14}$$

where $\beta_h > 0$. Note that in the above equation, the sign of a hidden unit's output weight, rather than its value, is used. This variation is based on Anderson's empirical results that the algorithm is more robust if only the sign of the weight is used rather than its value.

5.2 Learning in ASN

The ASN is a map from input to output space, denoted $F_{\mathbf{p}}(\mathbf{x})$. Here, $\mathbf{p}$ is the vector of all the weights in the network, which includes the centers and spreads of all antecedent and consequent labels used in the fuzzy rules. The intent of computing F is to maximize v, so that the system ends up in a good state and avoids failure. Hence, v is the objective function which needs to be maximized as a function of $\mathbf{p}$, given the state. This can be done by gradient descent, which estimates the derivative $\partial v / \partial p$, and uses the learning rule

$$\Delta p = \eta \frac{\partial v}{\partial p} = \eta \frac{\partial v}{\partial F} \frac{\partial F}{\partial p} \tag{15}$$

to adjust the parameter values. To do this, we need the two derivatives on the right hand side, which in general, will depend on the state.

Even though F is directly dependent on $\mathbf{p}$, the dependence of v on F is quite indirect. Each application of the force F is state-specific, and the new state depends in a complicated way on the dynamics of the plant. In addition, the transfer function of the AEN has to be taken into account to

see how the change in state affects v. Since part of this is unknown, and part of it is computationally complex, we have made the approximation that $\partial v/\partial F$ can be computed by the instantaneous difference ratio

$$\frac{\partial v}{\partial F} \approx \frac{dv}{dF} \approx \frac{v(t) - v(t-1)}{F(t) - F(t-1)} \tag{16}$$

Since this ignores the change in state between successive time steps, it is a very crude estimator of the derivative. We will therefore only use its sign, and not its magnitude. Of course, the existence of the derivative is an implicit assumption as well.

The other term $\partial F/\partial p$ is much more tractable. Since F is known and differentiable, a few applications of the chain rule through the 5 layers of the ASN give the following set of learning rules. In what follows, $\mathrm{Con}(R_j)$ and $\mathrm{Ant}(R_j)$ are the consequent label and antecedent labels used by Rule j. A label V is parameterized by p_V, which may be one of center, left spread or right spread.

For consequent labels V with parameters p_V, with z standing for μ^{-1}, the action F is linear in p_V, but nonlinear in w_i. Substituting for z_i using (7), and differentiating

$$F = \frac{\sum_r w_r z_r}{\sum_r w_r} \tag{17}$$

$$z_V(w_r) = c_V + \frac{1}{2}(s_{VR} - s_{VL})(1 - w_r) \tag{18}$$

$$\frac{\partial F}{\partial p_V} = \frac{1}{\sum_i w_i} \sum_{V=Con(R_j)} w_j \frac{\partial z_V}{\partial p_V} \tag{19}$$

$$\frac{\partial z_V}{\partial c_V} = 1 \tag{20}$$

$$\frac{\partial z_V}{\partial s_{VR}} = = \frac{1}{2}(1 - w_r) \tag{21}$$

$$\frac{\partial z_V}{\partial s_{VL}} = -\frac{1}{2}(1 - w_r) \tag{22}$$

These derivatives can be combined to compute $\frac{\partial F}{\partial p_V}$. If only consequent labels are to be tuned, this is all that needs to be calculated. In many problems, this may be sufficient as well, since some error in the specification of antecedent labels can be compensated for by modifying the consequent labels.

For antecedent labels, the calculations proceed similarly. The action depends on the degrees w_r, which in turn depend on the membership degrees μ_i generated in layer 2.

$$\frac{\partial F}{\partial w_r} = \frac{w_r z(w_r) + z'(w_r) - F}{\sum_i w_i} \tag{23}$$

$$\frac{\partial w_r}{\partial \mu_j} = \frac{e^{-k\mu_j}(1 + k(w_r - \mu_j))}{\sum_i e^{-k\mu_i}} \tag{24}$$

$$\frac{\partial F}{\partial \mu_V} = \sum_{V \in Ant(R_k)} \frac{\partial F}{\partial w_k} \frac{\partial w_k}{\partial \mu_V} \tag{25}$$

where $z'(w_r)$ is the derivative with respect to w_r.

These are the variables controlled by the parameters of the antecedent labels.

$$\frac{\partial F}{\partial p_V} = \frac{\partial F}{\partial \mu_V} \frac{\partial \mu_V}{\partial p_V} \tag{26}$$

$$\frac{\partial v}{\partial F} \approx \text{sgn}\left(\frac{v(t) - v(t-1)}{F(t) - F(t-1)}\right) \tag{27}$$

The above derivatives can now be combined to get the gradient.

$$\frac{\partial v}{\partial p_V} = \frac{\partial v}{\partial F} \frac{\partial F}{\partial p_V} \tag{28}$$

An appropriate multiplicative learning rate factor is used with this estimation of the gradient. This consists of the *perturbation* $s(t)$ computed by the Stochastic Action Modifier, and the internal reinforcement $\hat{r}$ generated by the AEN, in addition to a constant η, which is set to a small positive value. The reason for using $s(t)\hat{r}(t)$ as a learning factor is that if a large perturbation results in a good action, then there should be an extra reward given to the weights, since probabilistic search has really helped the system in this case. Conversely, if the large random deviation is not beneficial, then it should have minimal effect on the weights.

Since we are interested in *maximizing* v,

$$\Delta p_V(t) = \eta s(t)\hat{r}(t)\frac{\partial v}{\partial p_V} \tag{29}$$

is the learning equation. The derivatives can be computed locally by each node after receiving relevant values backpropagated through the network.

The only nodes whose weights will change are the ones in layer 2 and 4. All other edges have weights fixed at 1.

A word about the existence of these derivatives. If the $\mu()$ used in layer 2 are differentiable everywhere, then all the relevant derivatives will exist. However, for triangular membership functions, the derivative does not exist at three points, since the two limits are not equal at these points. The formally rigorous way to handle this is to consider the convex combination of all the gradients at the singular point, and to pick the one direction from this set that benefits the optimization algorithm most. A heuristic approximation to this scheme is to use an average of the two limits for the derivative at the singular points. We have chosen the simpler heuristic approach. Note that such a problem does not arise in layer 4 functions, since the LMOM method to compute $\mu^{-1}()$ gives a differentiable function, even if the corresponding μ is triangular in shape.

The other potential problem with derivatives in gradient descent methods is flat spots. When $\frac{\partial F}{\partial p_V}$ is 0 because the inputs lie outside the range of μ_V, then no learning will occur for p_V. Strictly speaking, this is reasonable since V played no role in determining the action for this particular input. However, if the input data is confined to a portion of the input space such that V does not play any role at all, then the parameters controlling V will not be modified. In other words, the system will fail to generalize over parts of input space where there is little or no data available. This problem is partially avoided by using the Stochastic Action Modifier, which randomly perturbs the action performed so that the state trajectory of the system will not remain confined to a region of small volume. In our experiments, we have also used random starting configurations after a failure occurs. This removes sensitive dependence of the learning system on initial conditions.

Slow learning may also occur because the process is caught in a narrow ravine with a gradually sloping bottom (as is known to happen with gradient descent methods in neural networks). This can be avoided by use of a momentum term [24], or some sort of linesearch technique to determine the optimal step size at each point [15]. We have not used any of these methods in our simulations because we did not encounter prohibitively slow learning. However, since a standard gradient descent is being used, any of these variations and additions to speed it up can always be used.

6 The Cart-Pole Balancing Problem

We now apply the GARIC approach to solve an interesting control problem. In this problem a pole is hinged to a motor-driven cart which moves on rail tracks to its right or its left. The pole has only one degree of freedom (rotation about the hinge point). The primary control tasks are to keep the pole vertically balanced and keep the cart within the rail track boundaries.

Four state variables are used to describe the system status, and one variable represents the force applied to the cart. These are:

- x: horizontal position of the cart;
- $\dot{x}$: velocity of the cart;
- θ: angle of the pole with respect to the vertical line;
- $\dot{\theta}$: angular velocity of pole θ;
- f: force applied to the cart.

The dynamics of the cart-pole system are modeled by the following non-linear differential equations [4]:

$$\ddot{\theta} = \frac{g \sin\theta + \cos\theta[\frac{-f - ml\dot{\theta}^2 \sin\theta + \mu_c sgn(\dot{x})}{m_c + m}] - \frac{\mu_p \dot{\theta}}{ml}}{l[\frac{4}{3} - \frac{m\cos^2\theta}{m_c + m}]}$$

$$\ddot{x} = \frac{f + ml[\dot{\theta}^2 \sin\theta - \ddot{\theta}\cos\theta] - \mu_c sgn(\dot{x})}{m_c + m}$$

where g is the acceleration due to gravity, m_c is the mass of the cart, m is the mass of the pole, l is the half-pole length, μ_c is the coefficient of friction of cart on track, and μ_p is the coefficient of friction of pole on cart. These equations were simulated by the Euler method, which uses an approximation to the above equations, and a time-step of 20 msec.

We assume that a failure happens when $|\theta| > 12$ degrees or $|x| > 2.4$ meters. However, we later show that the system learns even when the these two bounds are tightened. Also, we assume that the equations of motion of the cart-pole system are not known to the controller and only a vector describing the cart-pole system's state at each time step is known. In other

Table 1: The membership functions: 14 labels for the antecedent and 9 labels in the consequent

Label	Center	Left spread	Right spread	Label	Center	Left spread	Right spread
PO1	0.3	0.3	-1	PL	20.0	5.0	-1.0
ZE1	0.0	0.3	0.3	PM	10.0	5.0	6.0
NE1	-0.3	-1	0.3	PS	5.0	4.0	5.0
VS1	0.0	0.05	0.05	PVS	1.0	1.0	1.0
PO2	1.0	1.0	-1.0	NL	-20.0	-1.0	5.0
ZE2	0.0	1.0	1.0	NM	-10.0	6.0	5.0
NE2	-1.0	-1.0	1.0	NS	-5.0	5.0	4.0
VS2	0.0	0.1	0.1	NVS	-1.0	1.0	1.0
PO3	0.5	0.5	-1.0	ZE	0.0	1.0	1.0
NE3	-0.5	-1.0	0.5				
PO4	1.0	1.0	-1.0				
NE4	-1.0	-1.0	1.0				
PS4	0.0	0.01	1.0				
NS4	0.0	1.0	0.01				

words, the cart-pole arrangement is treated as a black box by the learning system.

Figure 7 presents the GARIC architecture as it is applied to this problem. The AEN network has 4 input units, a bias input unit, 5 hidden units and an output unit. The input state vector is normalized, so that the pole and cart positions lie in the range $[0, 1]$. The velocities are also normalized, but they are not constrained to lie in any range. The 35 weights of this net are initialized randomly to values in $[-0.1, 0.1]$. The learning rate for these weights is fixed at 0.3. The external reinforcement (i.e., the failure signal r) is received by the AEN and used to calculate the internal reinforcement $\hat{r}$. The discount factor γ used in this calculation is 0.9.

6.1 The Action Selection Network

The fuzzy control rules used to balance the pole successfully are shown in Table 6.1 and explained below. These completely determine the width of

PO1	PO2	null	null	PL
PO1	ZE2	null	null	PM
PO1	NE2	null	null	ZE
ZE1	PO2	null	null	PS
ZE1	ZE2	null	null	ZE
ZE1	NE2	null	null	NS
NE1	PO2	null	null	ZE
NE1	ZE2	null	null	NM
NE1	NE2	null	null	NL
VS1	VS2	PO3	PO4	PS
VS1	VS2	PO3	PS4	PVS
VS1	VS2	NE3	NE4	NS
VS1	VS2	NE3	NS4	NVS

Figure 1: The 13 rules used with 7 labels for force.

each layer in the ASN. There are 4 inputs, 14 units in layer 2 (the number of antecedent labels), 13 units in layer 3 (the number of rules), 9 units in layer 4 (the number of consequent labels) and finally, one output unit to compute the force. The initial definitions of all the labels are also shown in the table. These directly translate into the initial weights of Layers 2 and 4 in the Action Selection Network.

The design of the rule base for this fuzzy controller follows the algorithm developed in [9, 11] which is based on a hierarchical process which considers the interaction of multiple goals.

As mentioned earlier, the rule base of a fuzzy controller consists of rules which are described using *linguistic variables*. As shown in Figure 8(a) and Figure 8(b), four labels are used here to linguistically define the value of the state variables: Positive (PO), Very Small (VS), Zero (ZE), and Negative (NE). Nine labels are used to linguistically define the force value recommended by each control rule: Positive Large (PL), Positive Medium (PM), Positive Small (PS), Positive Very Small (PVS), Zero (ZE), Negative Very Small (NVS), Negative Small (NS), Negative Medium (NM), and Negative Large (NL). The forward calculations in this network are based on fuzzy logic control as described earlier. Nine fuzzy control rules were written for

balancing the pole vertically and four control rules were used in positioning the cart at a specific location on the rail tracks [11]. These rules are shown in Table 6.1. In Figure 7, the presence of a link between an input unit j and a unit i in the hidden layer indicates that the linguistic value of the input corresponding to unit j is used as a precondition in rule i. The first nine rules, corresponding to the hidden layer units 1 to 9, are rules with two preconditions (i.e., θ, and $\dot{\theta}$). The rules 10 through 13 have four preconditions representing the linguistic values of θ, $\dot{\theta}$, x, and $\dot{x}$.

For any particular control problem using the GARIC architecture, the fuzzy rules and their initial shapes and definitions need to be set up. We have used triangular membership functions for all antecedent and consequent labels. This choice is general enough to be applicable to many other problems besides cart-pole balancing. There are 13 rules for this 4-input system, and they use 23 linguistic labels in all. The spreads of a fuzzy membership function lie in the range $(0, \infty)$. If a spread is ∞, this parameter will not be changed during learning, and the defuzzification procedure (LMOM) will work by inverting the non-constant portion. In addition, the softmin parameter k is set at a value of 10, and the learning rate η is 0.01.

The labels and rule descriptions are presented in Figure 8. Given the rule base, the parameters may be thought of as a means of controlling the meaning of the linguistic terms. When the parameters change, this meaning is being tuned to be consistent with the rules, such that good performance results. In fact, performance is the only objective criterion of "correctness" of the label definitions, in the context of the fixed rulebase.

6.2 Results

A trial in our experiments refers to starting with the cart-pole system set to an initial state and ending with the appearance of a failure signal or successful control of the system for an extended period[‡]. The default parameters for the simulations are: half-pole length .5 m; Pole mass 0.1 Kg; Cart mass 1.0 Kg; learning rate in the consequents .001. The starting configuration after each failure was varied in numerous ways including randomly. The rules and starting label descriptions were varied by large amounts. The damages

[‡]We say that the system has learned to control the cart-pole if no failure is observed before 100 000 time steps. This time corresponds to about 33 minutes of real time.

to the labels which are the variations from labels original definitions, as well as changes in the parameters, are described with each figure. A starting position of 0.1, for example, implies that all 4 state variables were set to 0.1 after each failure. A randomized start means that after each failure, the initial configuration (all 4 parameters) were independently and randomly chosen. In the graphs, each curve shows the value of a state variable and is in four pieces. The first and second pieces show this value for the first few time steps of the first and second trials respectively. If the trial lasted less than 300 time steps, then the entire trial is shown, but if not, only the first 300 time steps are shown. The third and fourth pieces of the curve show the first 300 and last 300 (from 99700 to 100000) time steps of the last (successful) trial, when the experiment was terminated. Of course, failure occurs whenever θ or x exceed their respective bounds.

Figure 10 shows the performance of the controller during the learning process. This is to clearly demonstrate how the membership functions are shifted to the correct place by learning. In this experiment, we shifted the center of the membership function for ZE by 5 N (this is shown in the figure's caption by ZE +5). The system learned to shift it back to about 0 as shown in Figure 9. This change is sufficient for success, given the robustness of the fuzzy inference process. Other labels were also shifted by about 1 N, which is minimal change. The start state was non-random. Modifications to all force labels are shown in Table 2. Figures 11, 12, 13, 14, 15, and 16 illustrate the performance of the learning system under different scenarios which are described in the figure captions.

6.2.1 Additional Experiments

Two additional sets of experiments were performed. In the first set, we varied the number of labels for force from 9 to 7 and redefined their membership functions as shown in Table 3. Figures 17, 18, 19, 20, 21, 22, and 23 show the results of further experiments using the new membership functions with the rules which are shown in table 4.

Further experiments were performed using 9 modified labels for force as shown in table 5. The following table summarizes the results of these runs.

Table 2: Force labels after learning

Label	Center	Left spread	Right spread
PL	19.89	5.10	-1.00
PM	8.25	5.84	5.16
PS	6.73	3.99	5.80
PVS	1.08	0.99	1.01
NL	-20.29	-1.00	4.71
NM	-9.72	5.69	5.30
NS	-7.28	6.14	2.85
NVS	-0.09	0.31	1.68
ZE	-0.18	1.86	0.14

Table 3: The membership functions for the 7 force labels in the consequent

Label	Center	Left spread	Right spread
PL	20.0	1.0	-1.0
PS	5.0	1.0	1.0
PVS	1.0	1.0	1.0
NL	-20.0	-1.0	1.0
NS	-5.0	1.0	1.0
NVS	-1.0	1.0	1.0
ZE	0.0	1.0	1.0

Table 4: The 13 rules used for set 1 with 7 labels for force.

PO1	PO2	null	null	PL
PO1	ZE2	null	null	PL
PO1	NE2	null	null	ZE
ZE1	PO2	null	null	PS
ZE1	ZE2	null	null	ZE
ZE1	NE2	null	null	NS
NE1	PO2	null	null	ZE
NE1	ZE2	null	null	NL
NE1	NE2	null	null	NL
VS1	VS2	PO3	PO4	PS
VS1	VS2	PO3	PS4	PVS
VS1	VS2	NE3	NE4	NS
VS1	VS2	NE3	NS4	NVS

Table 5: Revised membership functions for force

Label	Center	Left spread	Right spread
PL	150	100	-1
PM	90	120	0
PS	0	0	80
PVS	0	0	20
NL	-150	-1	100
NM	-90	0	120
NS	0	80	0
NVS	0	20	0
ZE	0	0.2	0.2

Experiment Description	No. of Trials to learn
ZE +5., start from 0., learn rate = 0.01	34
Same as above, learning rate = 0.1	89
Same as above, learning rate = 0.001	32
Same as above, learning rate = 0.0001	85
ZE(force) +5,+5,+5	33
ZE1 +0.2	0

7 Discussion

GARIC's architecture is similar to the structure proposed by Anderson [2], but the action selection network in our architecture is a synthesis of fuzzy logic control and neural networks. Using the structure of a fuzzy controller, Anderson's approach is extended to provide for continuous representation of the output value and inclusion of the human expert operator's control rules in the action selection network. It should be noted that Anderson's goal in [1] was to discover interesting patterns and strategy-learning schemes. Not much effort was spent on making the process learn faster. In our work, although we allow some of the strategy learning to occur automatically, we start from a knowledge base of fuzzy control rules and tune them by learning in the neural networks.

Also, the stochastic action modifier unit in GARIC has similarities to Gullapalli's method [13] although we use a completely different approach for defining the internal reinforcement. Lee and Berenji [17] and Lee [16] have used a single layer neural network which requires the identification of the trace functions for keeping track of the visited states and their evaluations. The generation of these trace functions is a difficult task in larger control problems. However, the approach suggested in GARIC does not use trace functions. The neural network representation of the fuzzy control rules in GARIC allows faster development and faster learning. Also, in the single layer model, only the generation of the output values were considered. The preconditions of the fuzzy control rules were left untouched. However, in GARIC, based on reinforcements received from the environment, both the preconditions and the conclusions of rules can be modified.

The ARIC and GARIC architectures both use external reinforcements to form internal evaluations of states and control actions. Also, they both use

internal reinforcements to guide the process of tuning the rules. However, GARIC extends the theory for using reinforcement learning in fuzzy control in many respects including:

- Learning is achieved by full integration of fuzzy inference into a feed-forward network, which can then adaptively improve performance by using gradient descent methods.
- The fuzzy memberships used in the definition of the labels are modified (tuned) globally in all the rules rather than being locally modified in each each individual rule.
- GARIC can compensate for inappropriate definitions of fuzzy membership functions in the antecedent of control rules. We showed this attribute by damaging the labels used in the antecedents and observing how the system can learn a new control policy to succeed. To the best of our knowledge, GARIC is the first architecture to do this.
- GARIC introduces a new conjunction operator in computing the rule strengths of fuzzy control rules.
- GARIC introduces a new localized mean of maximum (LMOM) method in combining the conclusions of several firing control rules.
- Only monotonic membership functions are used in ARIC. However, GARIC allows any type of differentiable membership functions to be used in construction of fuzzy logic controller.

8 Conclusions

With the GARIC architecture, We have proposed a new way of designing and tuning a fuzzy logic controller. The knowledge used by an experienced operator in controlling a process can now be modeled using approximate linguistic terms and later refined through the process of learning from experience. GARIC provides a well-balanced method for combining the qualitative knowledge of human experts in terms of symbolic rules and learning strength of the artificial neural networks. Therefore, we believe that this architecture is general enough for use in other rule-based systems which perform fuzzy logic inference.

References

[1] C. W. Anderson. *Learning and Problem Solving with Multilayer Connectionist Systems.* PhD thesis, University of Massachusetts, 1986.

[2] C. W. Anderson. Strategy learning with multilayer connectionist representation. Technical Report TR87-509.3, GTE Laboratories Inc., May 1988.

[3] R.C. Atkinson, G.H. Bower, and E.J. Crothers. *An introduction to mathematical learning theory.* Weily, New York, 1965.

[4] A. G. Barto, R. S. Sutton, and C. W. Anderson. Neuronlike adaptive elements that can solve difficult learning control problems. *IEEE Transactions on Systems, Man, and Cybernetics*, 13:834–846, 1983.

[5] A.G. Barto. Connectionist learning for control: An overview. COINS tech. Report 89-89, University of Massachusetts at Amherst, 1989.

[6] A.G. Barto and M.I. Jordan. Gradient following without backpropagation in layered networks. In *IEEE First Annual Conference on Neural Networks*, pages II629–II636, San Diego, CA, 1987.

[7] A.G. Barto, R.S. Sutton, and C.J.C.H. Watkins. Sequential Decision Problems and Neural Networks. In D.S. Touretzky, editor, *Advances in Neural Information Processing Systems 2*, pages 686–693. Morgan Kaufmann, San Mateo, CA, 1990.

[8] H. R. Berenji. Fuzzy logic controllers. In R. R. Yager and L.A. Zadeh, editors, *An Introduction to Fuzzy Logic Applications in Intelligent Systems*, pages 69–96. Kluwer Academic Publishers, 1991.

[9] H. R. Berenji, Y. Y. Chen, C. C. Lee, S. Murugesan, and J. S. Jang. An experiment-based comparative study of fuzzy logic control. In *American Control Conference*, Pittsburgh, 1989.

[10] H.R. Berenji. An architecture for designing fuzzy controllers using neural networks. *International Journal of Approximate Reasoning*, 6(2):267–292, February 1992.

[11] H.R. Berenji, Y.Y. Chen, C.C. Lee, J.S. Jang, and S. Murugesan. A hierarchical approach to designing approximate reasoning-based controllers for dynamic physical systems. In *Sixth Conference on Uncertainty in Artificial Intelligence*, pages 362–369, 1990.

[12] J. A. Bernard. Use of rule-based system for process control. *IEEE Control Systems Magazine*, 8, no. 5:3–13, 1988.

[13] V. Gullapalli. A stochastic reinforcement learning algorithm for learning real-valued functions. *Neural Networks*, 3:671–692, 1990.

[14] Y. Kasai and Y. Morimoto. Electronically controlled continuously variable transmission. In *Int. Congress on Transportation Electronics*, Dearborn, Michigan, 1988.

[15] Alan Kramer and A. Sangiovanni-Vincentelli. Efficient parallel learning algorithms for neural networks. In *Advances in Neural Information Processing Systems*, volume 1, pages 40–48. Morgan Kaufmann, 1989.

[16] C.C. Lee. Self-learning rule-based controller employing approximate reasoning and neural-net concepts. *Int. Journal of Intelligent Systems*, 1990.

[17] C.C. Lee and H.R. Berenji. An intelligent controller based on approximate reasoning and reinforcement learning. In *Proc. of IEEE Int. Symposium on Intelligent Control*, Albany, NY, 1989.

[18] J.M. Mendel and R.W. McLaren. Reinforcement-learning control and pattern recognition systems. In J.M. Mendel and K.S. Fu, editors, *Adaptive, learning and pattern recognition systems: Theory and applications*, pages 287–318. Academic Press, New York, 1970.

[19] D. Michie and R. A. Chambers. Boxes: An experiment in adaptive control. In J.T. Tou and R. H. Wilcox, editors, *Machine Intelligence*, volume 2, pages 137–152. Oliver and Boyd, Edinburgh, 1968.

[20] John Moody and C. J. Darken. Fast learning in networks of locally-tuned processing units. *Neural Computation*, 1:282–294, 1989.

[21] K. Narendra and S. lakshmivarahan. Learning automata - a critique. *Journal of Cybernetics, and Information Science*, 1:53–65, 1977.

[22] K. Narendra and M.A.L. Thathachar. *Learning Automata: An Introduction.* Prentice Hall, Englewood, Cliffs, N.J, 1989.

[23] T. J. Procyk and E. H. Mamdani. A linguistic self-organizing process controller. *Automatica*, 15(1):15–30, 1979.

[24] D. Rumelhart, G. Hinton, and R.J. Williams. Learning internal representations by error propagation. In D. Rumelhart and J. McCelland, editors, *Parallel Distributes Processing*, pages 318–362. MIT Press, Cambridge, MA, 1986.

[25] A. L. Samuel. Some studies in machine learning using the game of checkers. Journal of r & d, IBM, 1959.

[26] R.S. Sutton. *Temporal Credit Assignment in Reinforcement Learning.* PhD thesis, University of Massachusetts, 1984.

[27] R.S. Sutton. Learning to predict by the methods of temporal differences. *Machine Learning*, 3:9–44, 1988.

[28] M.L Tsetlin. *Automaton Theory and Modeling of Biological Systems.* Academic Press, New York, 1973.

[29] B. Widrow and S.D. Stearns. *Adaptive Signal Processing.* Prentice-Hall, Englewood Cliffs, NJ, 1985.

[30] R. R. Yager. An alternative procedure for the calculation of fuzzy logic controller values. *Journal of Japanese Society of Fuzzy Technology (SOFT)*, 3(4):736–746, 1991.

[31] S. Yasunobu and S. Miyamoto. Automatic train operation by predictive fuzzy control. In M. Sugeno, editor, *Industrial Applications of Fuzzy Control*, pages 1–18. North-Holland, Amsterdam, 1985.

[32] L. A. Zadeh. Fuzzy sets. *Information and Control*, 8:338–353, 1965.

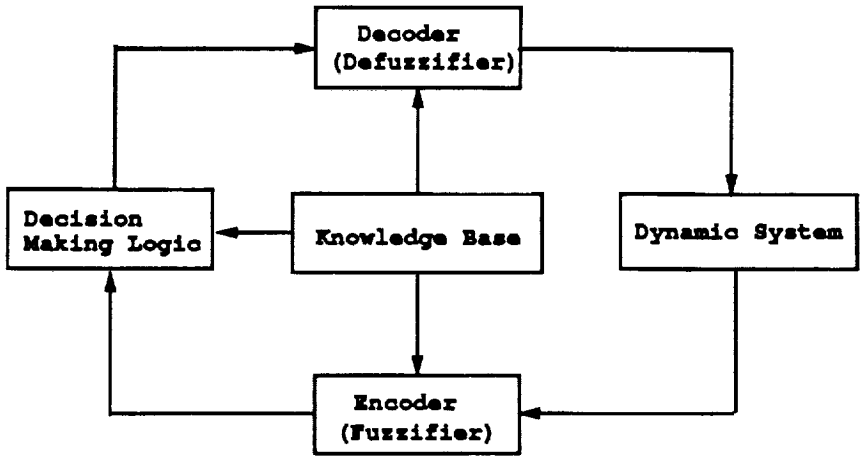

Figure 1: A simple architecture of a fuzzy logic controller

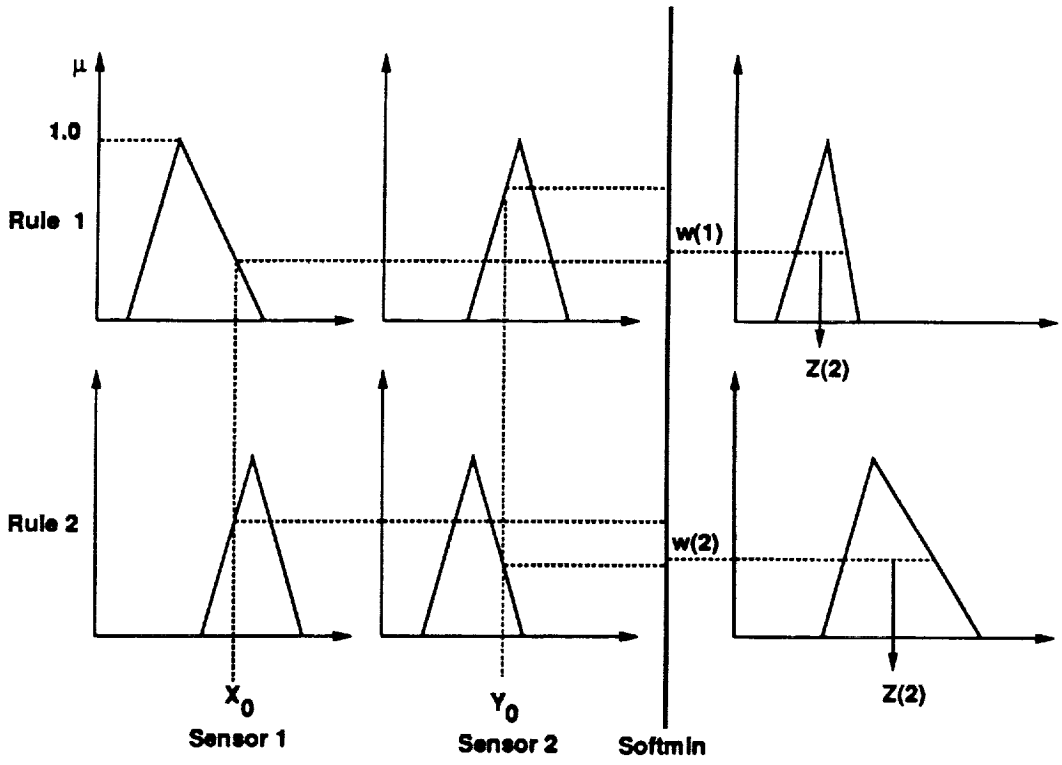

Figure 2: Fuzzification of antecedents using softmin.

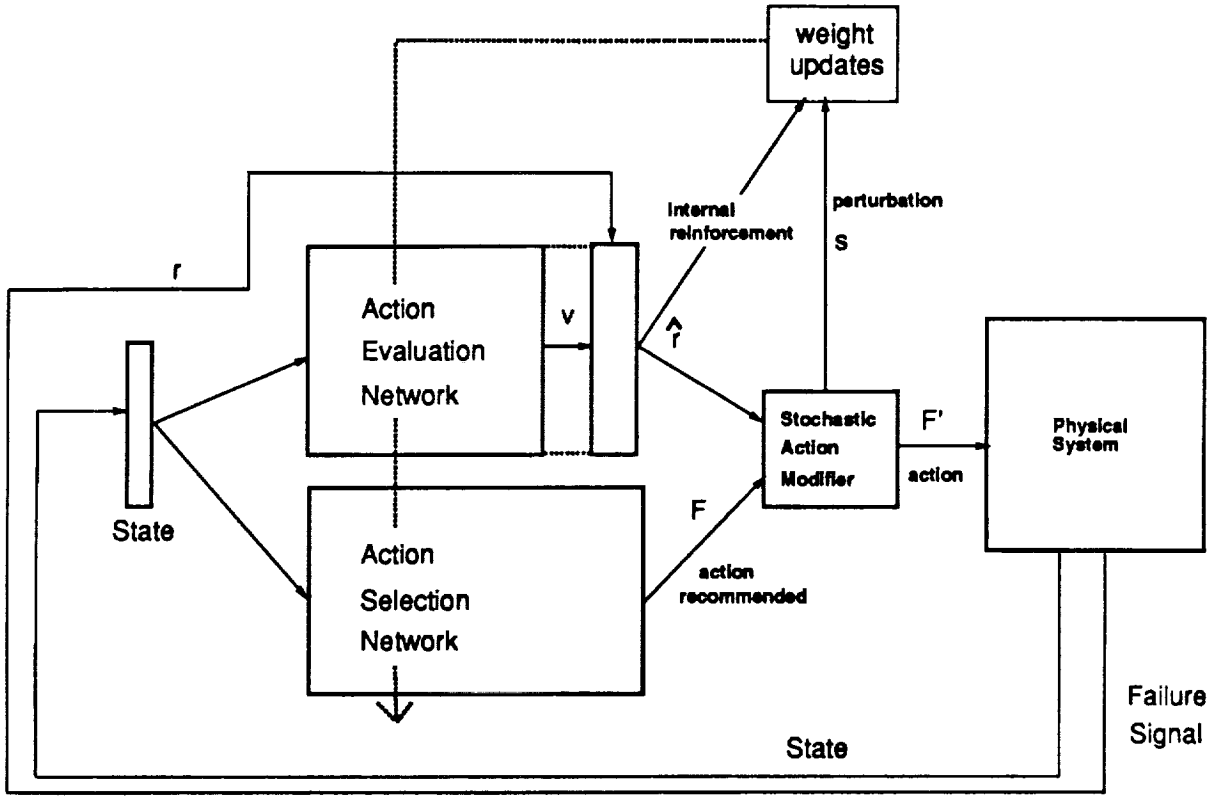

Figure 3: The Architecture of GARIC.

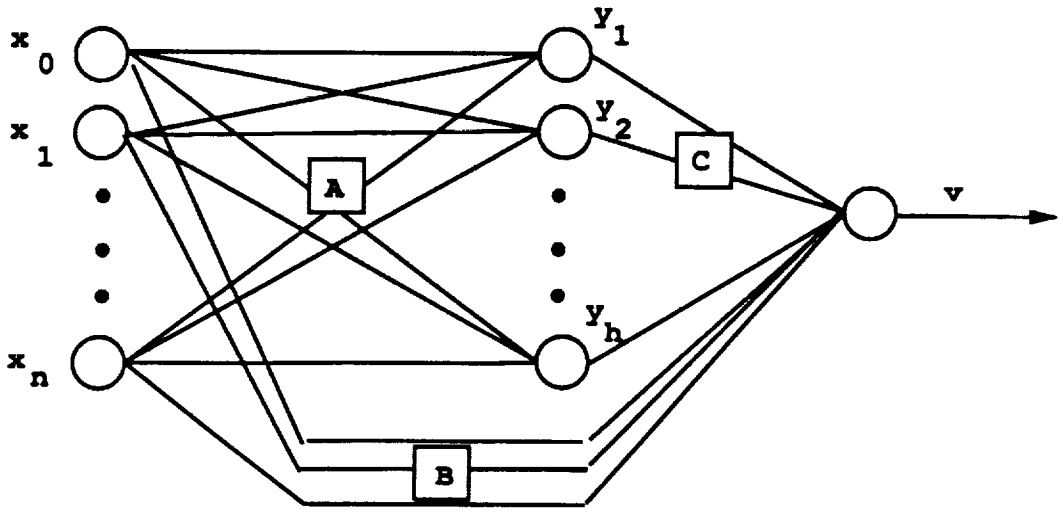

Figure 4: The Action Evaluation Network

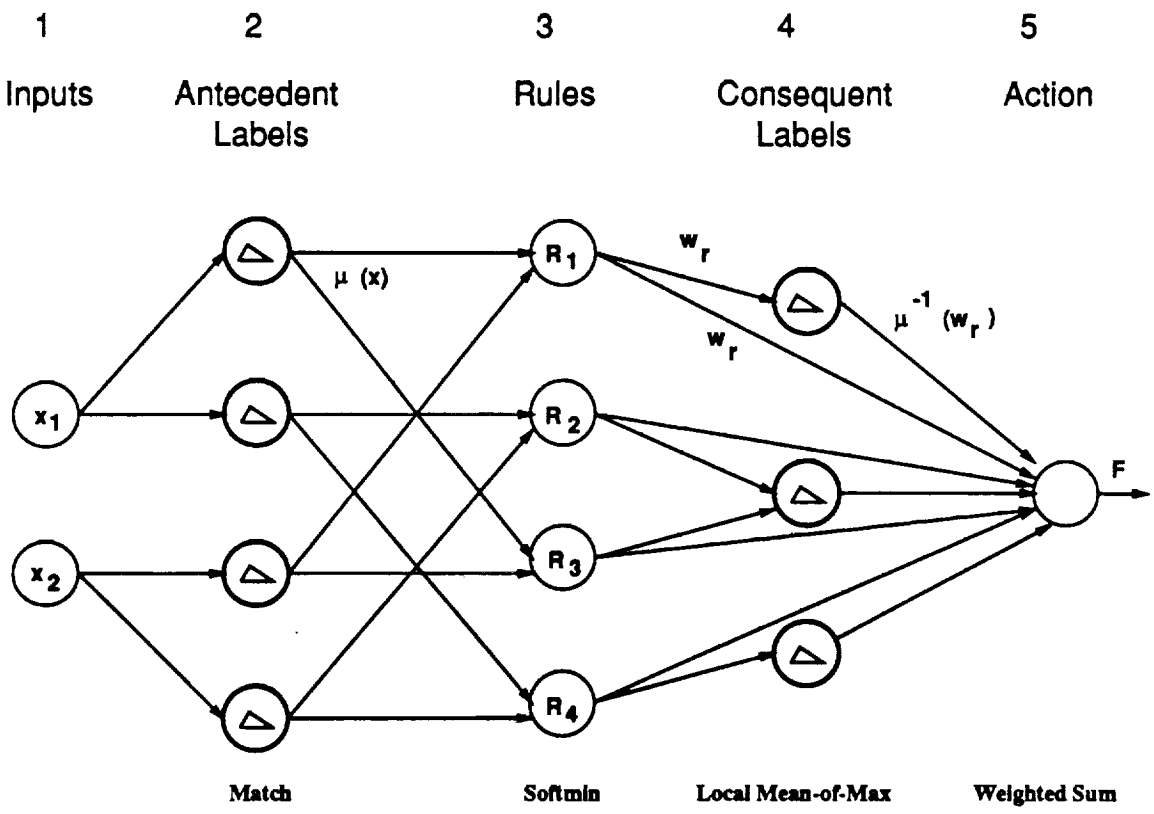

Figure 5: The Action Selection Network

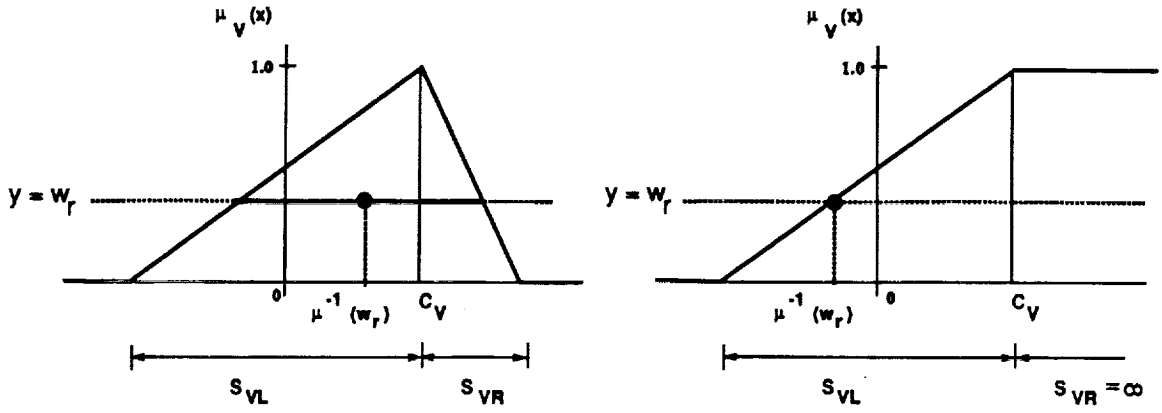

Figure 6: Local Mean-of-Maximum uses the Centroid of the line segment at height w_r intercepted by the fuzzy membership function, if it is non-monotonic

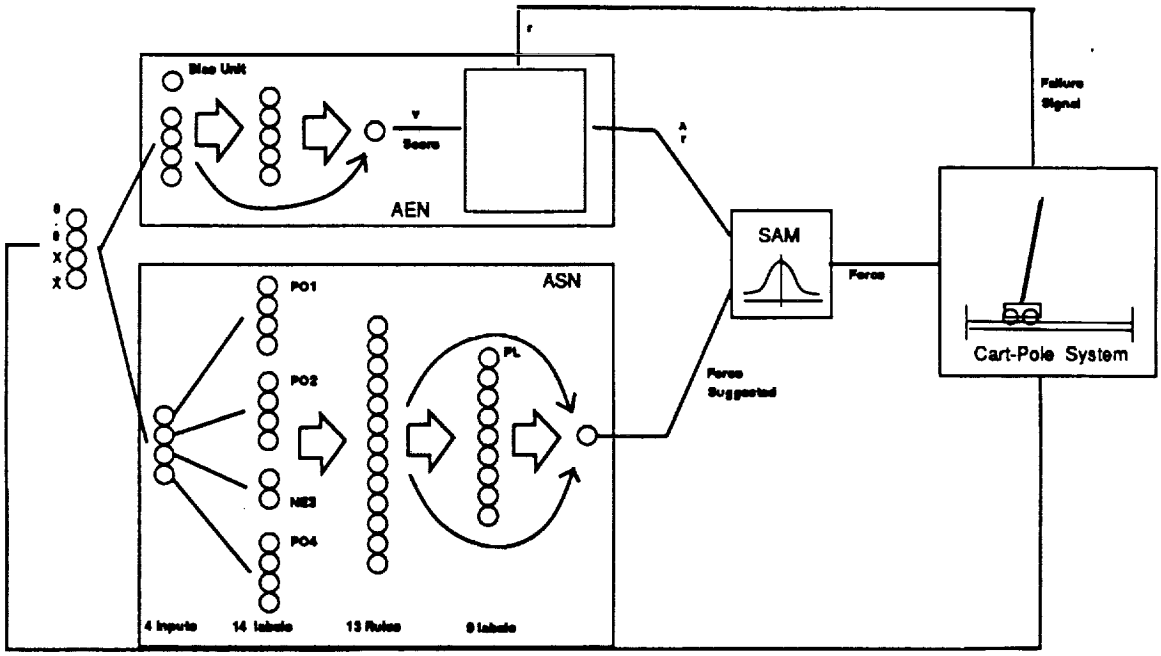

Figure 7: GARIC applied to cart pole balancing.

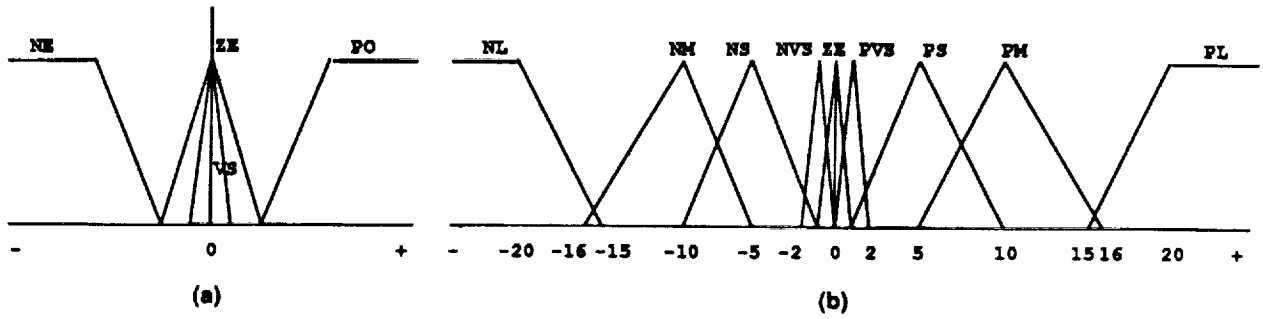

Figure 8: (a)- Four qualitative labels for $\theta, \dot{\theta}, x$, and $\dot{x}$, (b)- Nine qualitative labels for F.

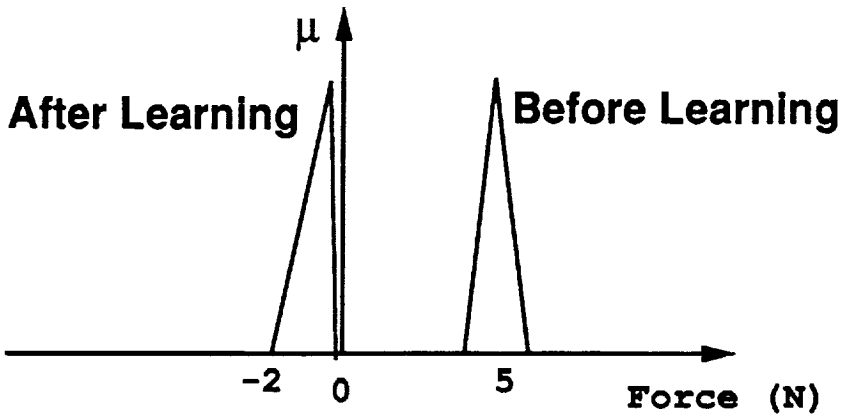

Figure 9: Learning to correct an inappropriate definition of a label's membership function.

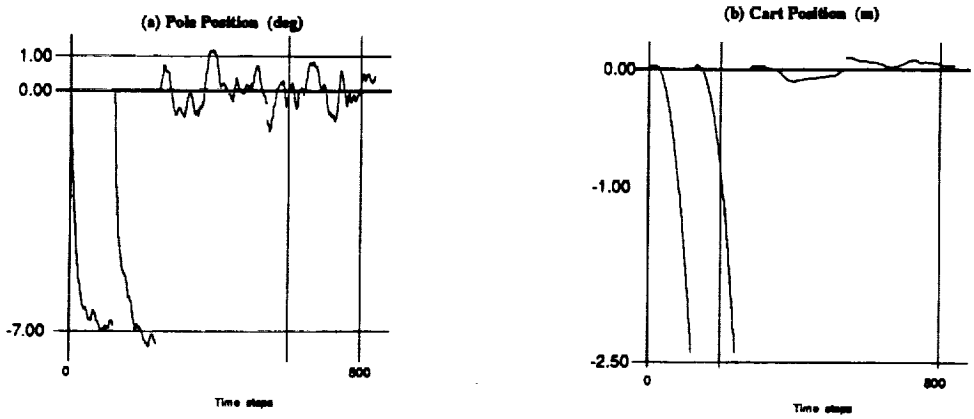

Figure 10: The ZE force label was set to +5. The system shifted it back to about 0, which is enough for success, given the robustness of the fuzzy inference process. Other labels were also shifted by about 1 N, which is minimal change. Start state was non-random. The system learned in 322 trials.

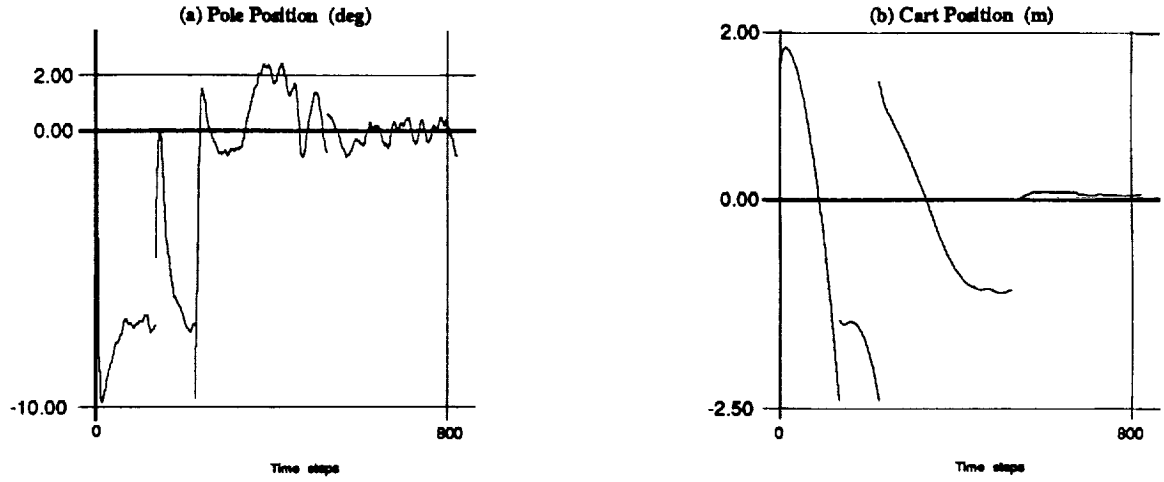

Figure 11: Here the starts are randomized. Other parameters are the same as abovare required to learn but not much more. Again, the system brings back the label from 5 to near 0. The system learned in 367 trials.

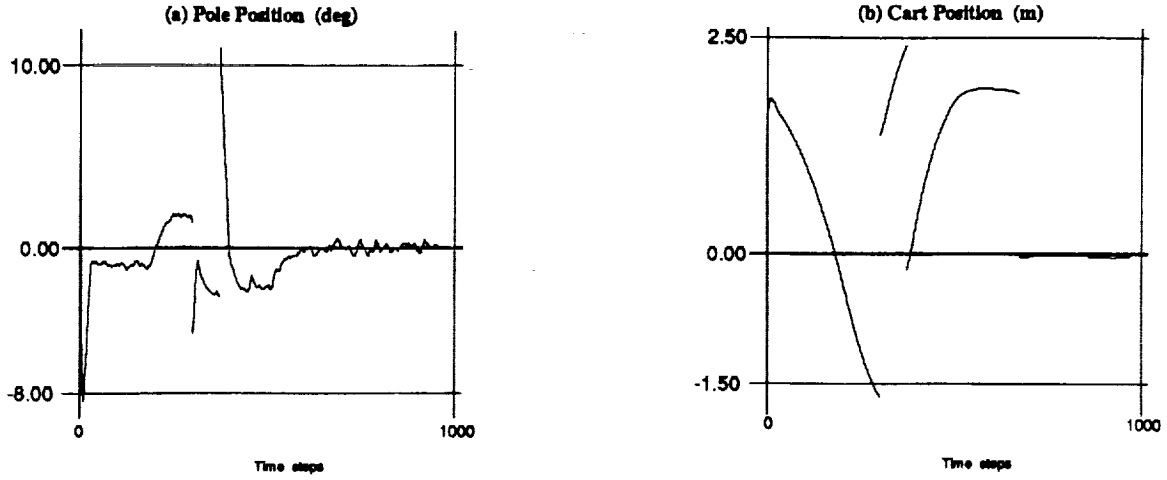

Figure 12: Antecedents change: ZE1 +0.2, ZE2 -0.4, PO3 -0.1, NS4 -0.1, ConsequenStart: randomized. The consequents are all adjusted by small amounts. ZE is brought back again because it is the most critical label in some sense. If this label is not correct, balancing is impossible even if the system has started from a good state. Here, the system learned in 312 trials.

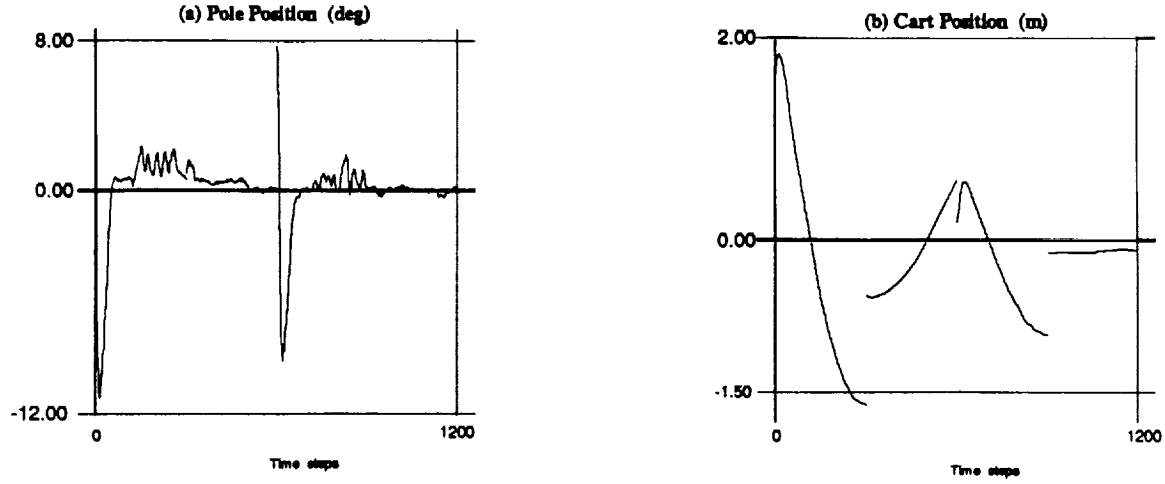

Figure 13: More damage in addition to above. Antecedents changed : ZE1 -0.2, ZE2 -0.4, PO3 -0.1, NE3 -1.0,0,+1 PO4 +0.3,+0.3,0 PS4 +0.3 NS4 -0.1. Consequents changed : PL +5, PM +3, PS +2, NM +1, ZE -2. Start : randomized. Again, ZE is corrected and the others are shifted by comparatively smaller amounts. PM is brought back near to its original value. In general, there is less pressure to correct the less-frequently used labels (such as PL), since once the system is in a good area of state-space, it can completely avoid using PL after it has made the important repairs. Learning took 550 trials.

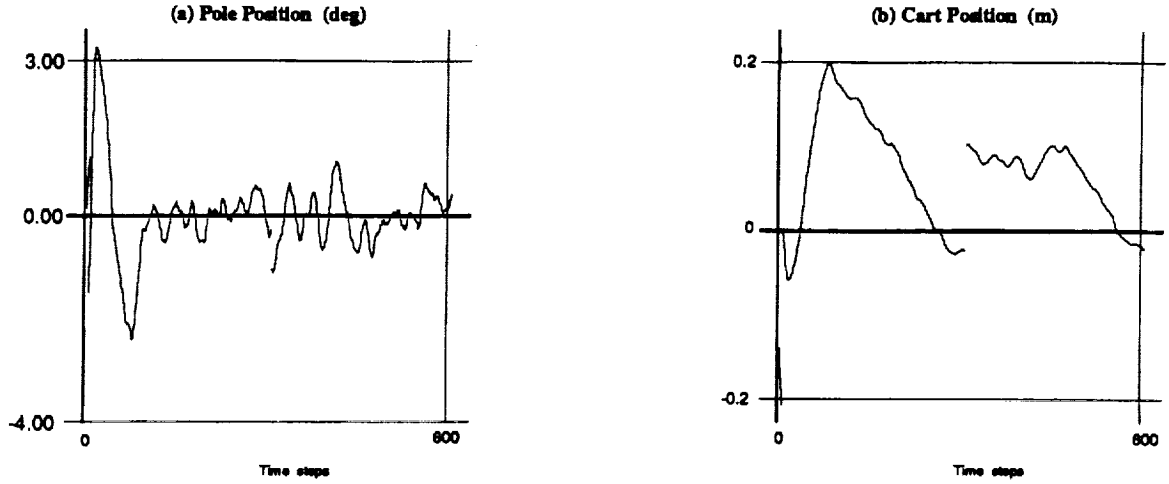

Figure 14: Parameter damage : maximum pole position = 0.1 rad and maximum cart position = 0.2 m which have been reduced from 0.2 rad and 2.4 m respectively. Start was randomized. Learning took 82 trials.

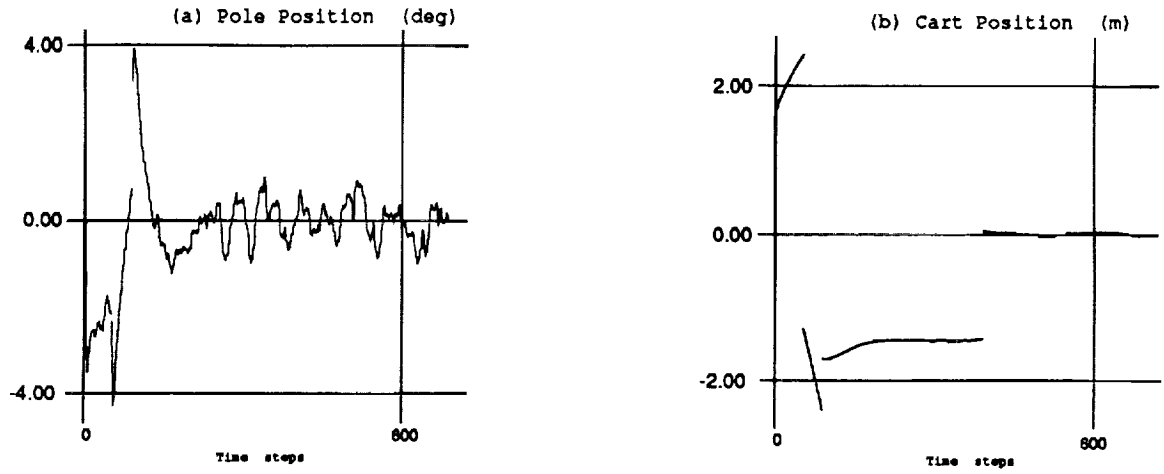

Figure 15: Pole half-length was reduced to 0.2 m from 0.5 m, and the start was randomized. Only 4 trials required for learning.

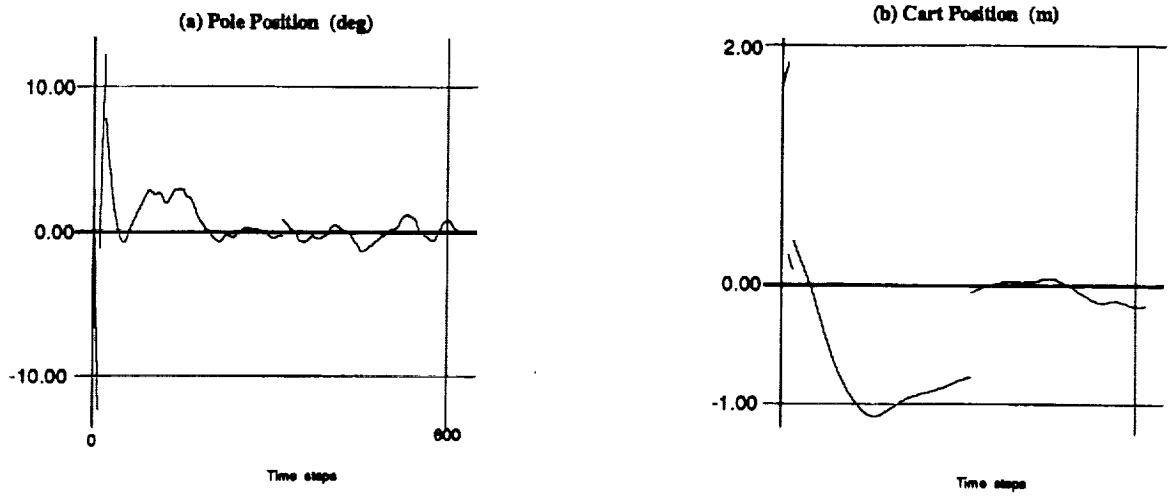

Figure 16: Cart mass was doubled to 2.0 kg. Random starting state. 2 trials were enough for success. The PL and NL shift their centers away by small amounts to compensate for the heavier cart. Similar effects are observed in other labels. The shifts are all small, since the effect is distributed over a large number of parameters.

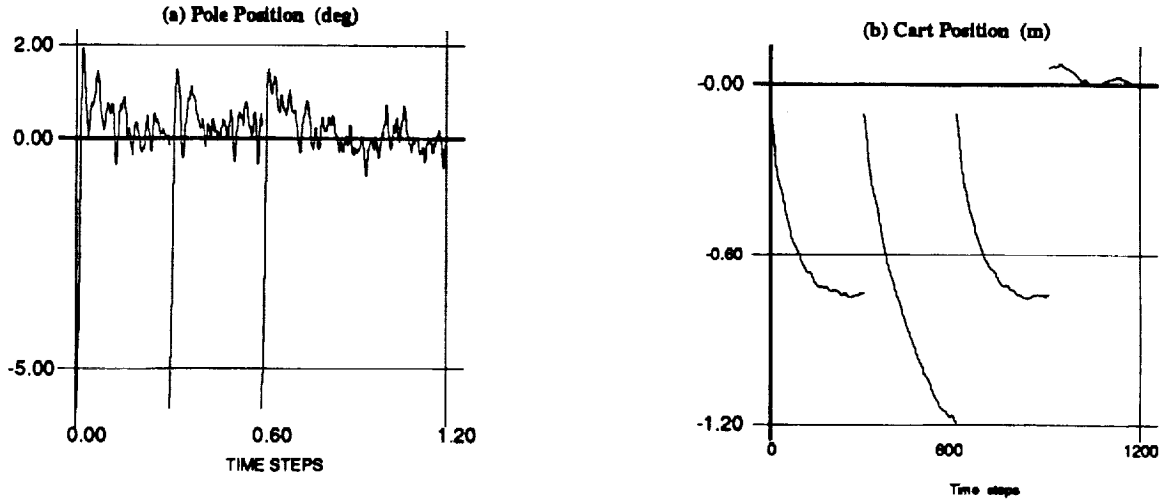

Figure 17: Damages of: PO4 (+0.4,+0.4,0), NE4 (+0.2,0,-0.2), PS4 (+0.5,0,0), NS4 (-0.5,0,0), starting state = 0.1. Learned in 2 trials.

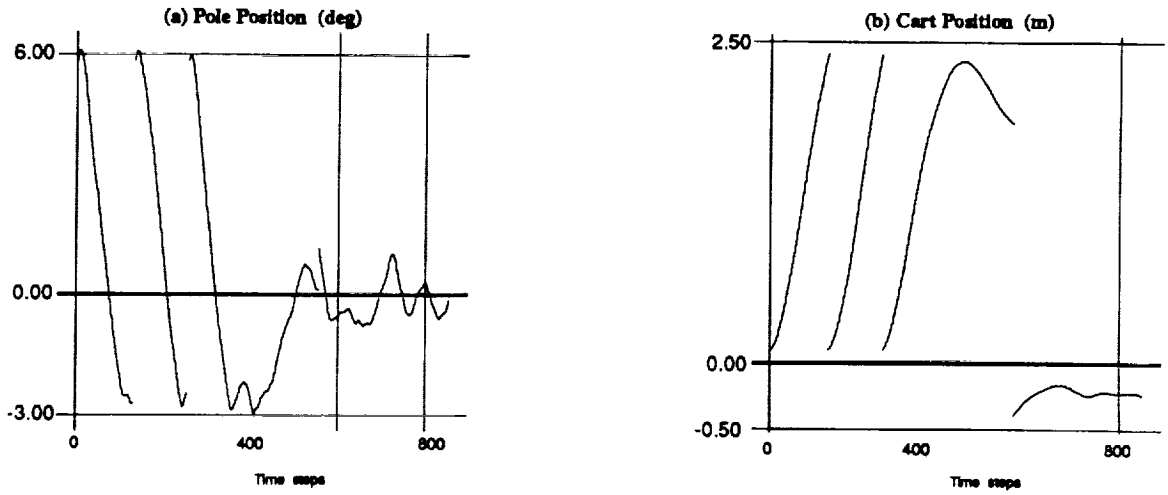

Figure 18: Cart mass = 4.3 kg (increased from 1 kg), starting position = 0.1, learning occurred in 3 trials.

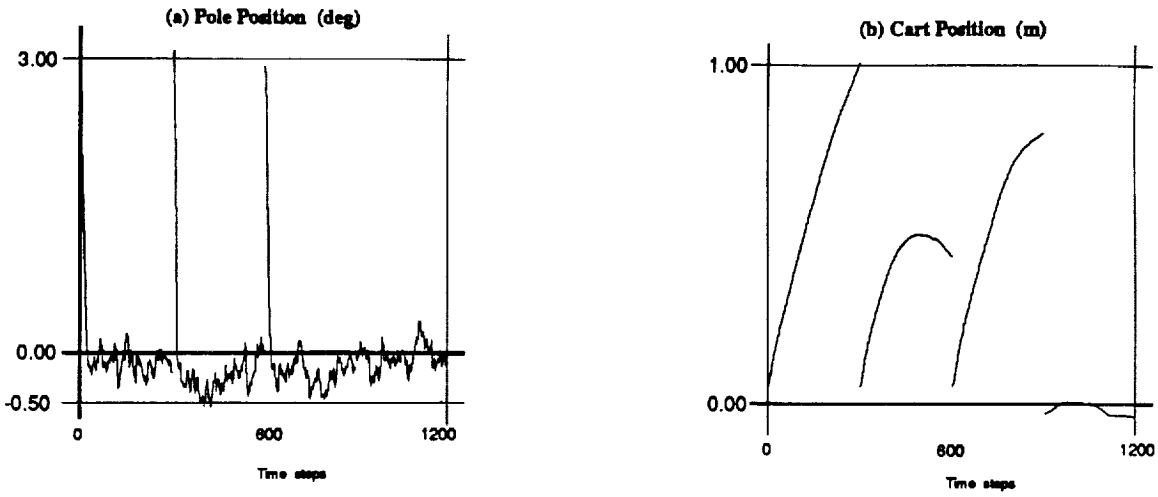

Figure 19: Damages to all labels of θ : (+0.2,+0.2,0),(+0.2,0,0),(-0.2,0,+0.2),(-0.1,0,0) to PO1,ZE1,NE1,VS1 respectively. Starting position = 0.05. It took 29 trials to succeed.

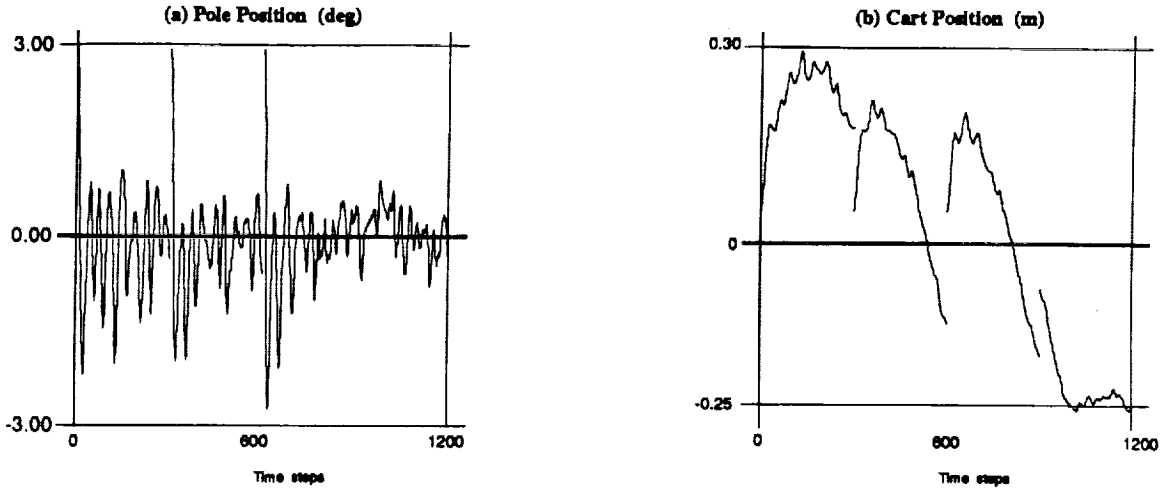

Figure 20: Damages to all labels of $\dot{\theta}$: (+0.4,+0.6,0),(+0.1,0,0),(-0.2,0,+0.4),(-0.1,+0.3,0) to PO2,ZE2,NE2,VS2 respectively. Starting position = 0.05. The learning time was 24 trials.

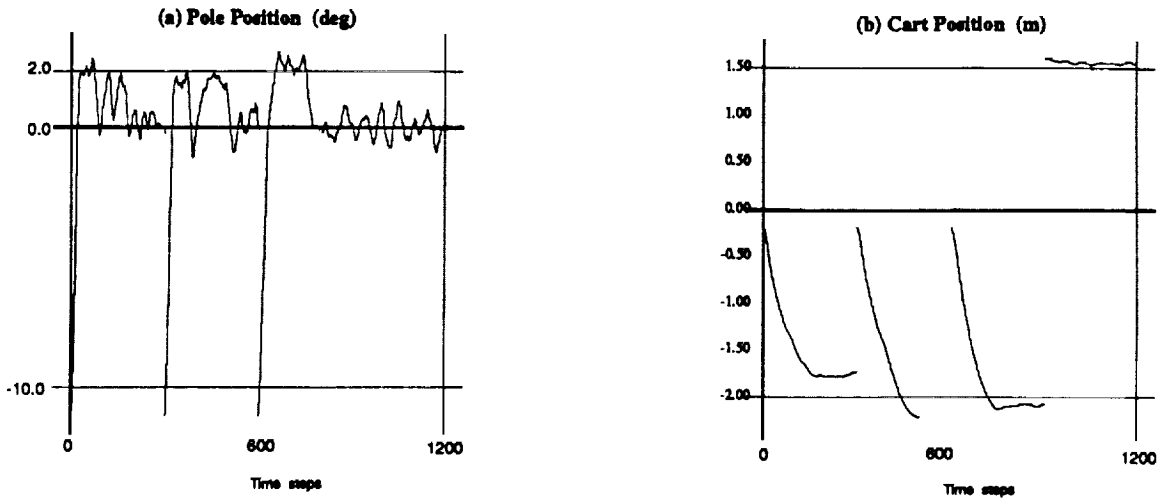

Figure 21: Damage to consequents = (-5,-1,0,5.5,1,0,0). Starting position = -0.19 (which is very close to failure). 136 trials were required.

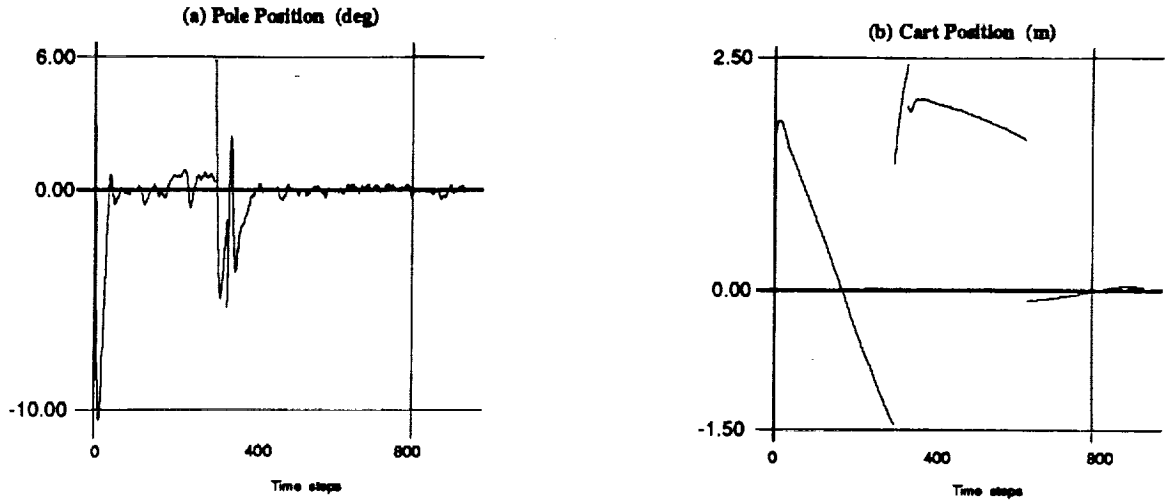

Figure 22: Major damage to consequents : PL +40.0, NL +5. Randomized starting positions. 15 trials were sufficient.

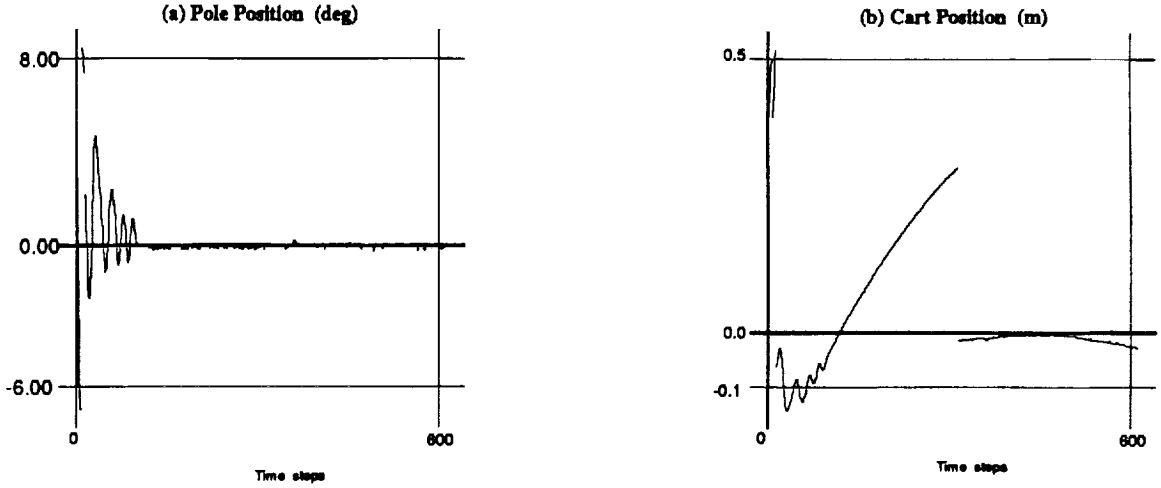

Figure 23: Max cart position = 0.5 (from 2.4). Randomized starts. Antecedent damage : ZE1 +0.3, ZE2 0.6, VS2 +0.2, PO3 -0.4, NE4 +0.5, PS4 +0.1, NS4 +0.3. Learned in 140 trials.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY *(Leave blank)*	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	Dates attached	

4. TITLE AND SUBTITLE

Titles/Authors - Attached

5. FUNDING NUMBERS

6. AUTHOR(S)

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

Code FIA - Artificial Intelligence Research Branch
Information Sciences Division

8. PERFORMING ORGANIZATION REPORT NUMBER

Attached

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)

Nasa/Ames Research Center

Moffett Field, CA. 94035-1000

10. SPONSORING/MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES

12a. DISTRIBUTION/AVAILABILITY STATEMENT

Available for Public Distribution

Peter Friedland 5/14/92 BRANCH CHIEF

12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*

Abstracts ATTACHED

14. SUBJECT TERMS			15. NUMBER OF PAGES
			16. PRICE CODE
17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT

PRECEDING PAGE BLANK NOT FILMED